

## Appendix A

```
package analysis;

5
import acme.*;
import java.util.*;
import java.io.*;
import java.awt.*;
10 import java.awt.event.*;
import javax.swing.*;

////////////////////////////////////
public class Analysis {
15
    // Temp for standalone analysis project. rundatastream.java
    public final static short TEMP = 7, OPTICS = 1 * 1024;

    public final static int NORMAL = 0, RAW = 1, DERIV1 = 2, DERIV2 = 3,
20         DERIV1RAW = 4, DERIV2RAW = 5, MELT_OPTICS = 6,
        MELT_TEMPERATURE = 7, MELT_DERIV1 = 8;
    public final static int MAX_CYCLES = 100;
    public final static int MAX_DYES = 4;
    public final static int MAX_SITES = 96;

25
    // Results
    public final static int PASS = 0;
    public final static int FAIL = 1;
    public final static int NO_RESULT = 2;    // eg, passive dye
30 public final static int ND = 3;           // Not Determined, IC invalid

    // Dye Types
    public final static int UNUSED = 0;
    public final static int ASSAY = 1;
35 public final static int INTERNAL_CONTROL = 2;
    public final static int QIC = 3;
    public final static int PASSIVE = 4;    // Historical but needed
    public final static int UNKNOWN = 5;   // Qual. Find conc. for this dye
    public final static int STANDARD = 6;  // Qual. Dye with known conc.

40
    // Site Designation
    public final static int SITE_UNKNOWN = 0;
    public final static int SITE_STANDARD = 1;

45
    // Data to use
    public final static int PRIMARY = 0;
```

```

public final static int D2 = 1;           // 2nd Derivative

// Analysis Type
public final static int QUALITATIVE = 0;
5 public final static int QUANTITATIVE = 1;

// Threshold mode
public final static int AUTO_THRESH = 0;
10 public final static int MAN_THRESH = 1;

public static boolean annotate = false;

// Setup, results...
Site site[];
15 private int analysisType;

// Num Sites
private int numSites;
20

// One per dye, site independent
// Primary: 0; 2D: 1
int dataType[] = new int[MAX_DYES];

25 // Following used for the standards curve, prakash 1/25/00
double dyeSlope[] = new double[MAX_DYES]; // m: mx+b
double dyeOffset[] = new double[MAX_DYES]; // b: mx+b
double linCC[] = new double[MAX_DYES];

30 // standardsLine[0-3][2]
// Each point is defined by (cycle, logb10(concentration))
public StdElement standardsLine[][] = new StdElement[MAX_DYES][2];
public static int stdChannel = 0;

35 // IC used: T, IC not used:F
private boolean useIC;
private int icDye;

// QIC used: T, QIC not used:F
40 private boolean useQIC;
private int qicDye;

// Threshold Mode (1 per dye)
private int threshMode[] = new int[MAX_DYES];
45

// Valid Cycle Number Range for all dyes

```

```

private float validMinCycle[] = new float[MAX_DYES];
private float validMaxCycle[] = new float[MAX_DYES];

// Cycle Number for noise sub and 3 sigma calculation.
5 boolean noise;
int baselineStartCycle[] = new int[MAX_DYES];
int baselineEndCycle[] = new int[MAX_DYES];

// StdDev baseline for auto threshold detect. User entered,
10 // one per dye.
private double stdDevBaseLine[] = new double[MAX_DYES];

// The Max stdDev for a given dye. one per dye
private float maxStdDev[] = new float[MAX_DYES];

15 // This is set to true only if all sites have a valid
// stdDev. Than only can you calculate the max.
private boolean maxStdDevValid[] = new boolean[MAX_DYES];

20 // BoxCar Averaging
private boolean boxCar;
private int boxCarWidth; // Note Min Value = 2

// Quantitative Analysis
25 public StdElement qtArr[][] = new StdElement[MAX_DYES][1];

// //////////////////////////////////////
// Keeps current settings, resets Data (and all calculated values from data)
30 // //////////////////////////////////////
public void resetData() {

    for(int s = 0; s < numSites; s++) {
        site[s].cycle = 0;
        35 site[s].control = false;
        site[s].meltPoints = 0;

        for(int d = 0; d < MAX_DYES; d++) {
            site[s].dye[d].tValid = false;
            40 site[s].dye[d].tCycle = 0f;
            site[s].dye[d].stdDevValid = false;
            site[s].dye[d].slope = 0.;
            site[s].dye[d].offset = 0.;
            site[s].noiseValid[d] = false;
            45 }
        }
    }
}

```

```

// qtArr = null;
StdElement a[] = new StdElement[1];

5   a[0] = new StdElement();

    // Site independent
    for(int d = 0; d < MAX_DYES; d++) {
10      maxStdDev[d] = 0f;
        maxStdDevValid[d] = false;

        qtArr[d] = null;
        qtArr[d] = a;    // Reset Quantation

15      standardsLine[d][0] = new StdElement();
        standardsLine[d][1] = new StdElement();
        dyeSlope[d] = 0.;
        dyeOffset[d] = 0.;
        linCC[d] = 0.;
20    }
  }

// //////////////////////////////////////
25 // Keeps current optics data, redoes all calculations. Eg. may be called
    // after changing Threshold mode from manual to auto.
    // //////////////////////////////////////
    public void recalc() {
        int s, cy;

30      //System.out.println("Analysis.recalc()");

        int c[] = new int[numSites];
        int meltCount[] = new int[numSites];

35      for(s = 0; s < numSites; s++) {
          c[s] = site[s].cycle;
          meltCount[s] = site[s].meltPoints;
        }

40      resetData();

        for(cy = 0; cy < c[0]; cy++) {
          for(s = 0; s < numSites; s++) {
45            addCycle(s, site[s].dye[0].rOptic[cy], site[s].dye[1].rOptic[cy],
                      site[s].dye[2].rOptic[cy], site[s].dye[3].rOptic[cy]);
          }
        }

```

```

    }
    }
}

5
// //////////////////////////////////////
public void setNumSites(int s) {
    if(s <= 0) {
        return;
10    }

    if(s < numSites) {
        for(int i = s; i < numSites; i++) {
            site[i] = null;
15        }
    }
    numSites = s;
}

20
// //////////////////////////////////////
public void addCycle(int s, short op0, short op1, short op2, short op3) {
    int c = site[s].cycle;

25    //System.out.println("addCycle Site " + s + " Op0 " + op0);

    site[s].dye[0].rOptic[c] = op0;
    site[s].dye[1].rOptic[c] = op1;
    site[s].dye[2].rOptic[c] = op2;
30    site[s].dye[3].rOptic[c] = op3;

    site[s].dye[0].pOptic[c] = op0;
    site[s].dye[1].pOptic[c] = op1;
    site[s].dye[2].pOptic[c] = op2;
35    site[s].dye[3].pOptic[c] = op3;

    processData(s);

    ++site[s].cycle;
40 }

// //////////////////////////////////////
public void addMelt(int s, short secs, int type, short value) {

45    //System.out.println("addMelt Site " + s + " sec " + secs + " type " + type + "
    value " + value);

```

```

site[s].meltPoints = secs;

switch(type) {
5 //case RunDataStream.OPTICS:
  case OPTICS:
    site[s].mOptic.set(secs, value);
    site[s].updateMeltDeriv();
    break;
10
  //case RunDataStream.TEMP:
  case TEMP:
    site[s].mTemp.set(secs, ((float)value / 100f));
    break;
15 }
}

// ////////////////////////////////////////
20 // 0=Ql, 1=Qn
public void setAnalysisType(int a) {
  analysisType = a;
}

// ////////////////////////////////////////
25 // To Manually set Threshold limit
// Call this once per dye
public void setTLimit(int d, float tl) {
30   for(int s = 0; s < numSites; s++) {
     site[s].dye[d].tLimit = tl;
   }
}

// ////////////////////////////////////////
35 // For testing quantation only.
// Call this once per dye
private void setTCycle(int s, int d, float tc) {
40   site[s].dye[d].tCycle = tc;
   site[s].dye[d].tValid = true;
}

// ////////////////////////////////////////
45 // 0=Auto, 1=Man

```

```

public void setTMode(int d, int tm) {
    threshMode[d] = tm;
}

```

5

```

// //////////////////////////////////////
// Conc. values for Quantitative analysis is set per site per dye
public void setConc(int s, int d, float conc) {
    site[s].dye[d].conc = conc;
}

```

10

```

// //////////////////////////////////////
// 0=Primary, 1=2D
public void setDataType(int d, int dt) {
    dataType[d] = dt;
}

```

15

```

// //////////////////////////////////////
// 0=UNKNOWN, 1=STANDARD
// In the GUI, SITE_UNKNOWN = 0 and SITE_STANDARD = 1
public void setSiteType(int s, int ty) {
    for(int d = 0; d < MAX_DYES; d++) {
        if(!((useIC && d == icDye) || (useQIC && d == qicDye))) {
            site[s].dye[d].dyeUsage = ty + 5;
        }
    }
}

```

20

25

30

```

// //////////////////////////////////////
// Unused/Std/Passive...
public void setDyeUsage(int s, int d, int du) {

```

35

```

    switch(du) {

        case INTERNAL_CONTROL:
            for(int si = 0; si < numSites; si++) {
                site[si].dye[d].dyeUsage = du;
            }

```

40

```

        useIC = true;
        icDye = d;

```

45

```

        break;
    }
}

```

```

case QIC:
    for(int si = 0; si < numSites; si++) {
        site[si].dye[d].dyeUsage = du;
    }

    useQIC = true;
    qicDye = d;

    break;
}

// ////////////////////////////////////////
// d=Dye, sd = standard dev. Set by User
public void setStdDevbaseline(int d, double sd) {
    stdDevBaseLine[d] = sd;
}

// ////////////////////////////////////////
// IC and Qic
public void setICCycle(int d, int min, int max) {
    validMinCycle[d] = (float)min;
    validMaxCycle[d] = (float)max;
}

// ////////////////////////////////////////
public void setNoiseSubtraction(boolean flag) {
    noise = flag;
}

// ////////////////////////////////////////
public void setBaselineCycle(int dye, int start, int end) {
    baselineStartCycle[dye] = start;
    baselineEndCycle[dye] = end;
}

// ////////////////////////////////////////
public void setBoxCarAvg(boolean flag, int width) {
    boxCar = flag;
    boxCarWidth = width;
}

```

```

    }

    // ////////////////////////////////////////
5   // Get Thresholds
    public float getTLimit(int s, int d) {
        //System.out.println("Analysis: getTLimit() " + site[s].dye[d].tLimit );
        return site[s].dye[d].tLimit;
    }

10

    // ////////////////////////////////////////
    public float getTCycle(int s, int d) {
        if (site[s].dye[d].tCycle < validMinCycle[d] || site[s].dye[d].tCycle >
15  validMaxCycle[d])
            return 0f;
        else
            return site[s].dye[d].tCycle;
    }

20

    // ////////////////////////////////////////
    public float getQICTCycle(int s, int d) {

25        int qicDye = getQICDye();
        float qicTCycle = getTCycle(s, qicDye);

        if (useQIC && (qicTCycle > 0f)) {
            if (d == qicDye) return qicTCycle;
            return (getTCycle(s,d) / qicTCycle);
30        }
        else
            return 0f;
    }

35

    // ////////////////////////////////////////
    public boolean getTValid(int s, int d) {
        return site[s].dye[d].tValid;
40    }

    // ////////////////////////////////////////
    public final double log10(double a) {
        if(a > 0.) {
45        return (Math.log(a) / Math.log(10.));
        }
    }

```

```

    else {
        return -9.5;
    }
}
5

// ////////////////////////////////////////
public final double log10(float a) {
    if(a > 0.) {
10        return (Math.log((double) a) / Math.log(10.));
    }
    else {
        return -9.5;
    }
15 }

// ////////////////////////////////////////
// Get Results
// ////////////////////////////////////////
20 public int getQLResult(int s, int d) {

    int du = site[s].dye[d].dyeUsage;

25 // Update IC
    if(useIC &&!site[s].control) {
        updateIC(s);
    }

30 if(du == UNUSED || du == PASSIVE) {
    site[s].dye[d].qlResult = NO_RESULT;

}
35 else if(useIC) {
    if(site[s].control) {
        site[s].dye[d].qlResult = site[s].dye[d].tValid ? PASS : FAIL;
    }
    else {
40     site[s].dye[d].qlResult = ND;
    }
}
else {
45     site[s].dye[d].qlResult = site[s].dye[d].tValid ? PASS : FAIL;
}

```

```

return site[s].dye[d].qlResult;
}

5 ///////////////////////////////////////////////////////////////////
// Update Internal Control Status
void updateIC(int s) {

10     if(site[s].dye[icDye].tValid) {

        // Also make sure it happened in the specified range
        if((site[s].dye[icDye].tCycle >= validMinCycle[icDye]) &&
            (site[s].dye[icDye].tCycle <= validMaxCycle[icDye])) {
            site[s].control = true;
15     }
        else {

            // Although .tValid, not in the range
            site[s].control = false;
20     }
        }
        else {
            site[s].control = false;
25     }
    }

    ///////////////////////////////////////////////////////////////////
    // Update Linear Correlation Coefficient
    ///////////////////////////////////////////////////////////////////
30 void updateCC(int d) {

        double yt, xt;
        double syy = 0., sxy = 0., sxx = 0., ay = 0., ax = 0.;
35
        if(qtArr[d].length < 2) {
            linCC[d] = 0.;

            return;
40     }

        for(int j = 0; j < qtArr[d].length; j++) {
            ax += qtArr[d][j].conc;
            ay += qtArr[d][j].avgTCycle;
45     }
    }

```

```

ax /= qtArr[d].length;
ay /= qtArr[d].length;

for(int j = 0; j < qtArr[d].length; j++) {
5   xt = qtArr[d][j].conc - ax;
   yt = qtArr[d][j].avgTCycle - ay;
   sxx += xt * xt;
   syy += yt * yt;
   sxy += xt * yt;
10  }

linCC[d] = sxy / (Math.sqrt(sxx * syy));
linCC[d] *= linCC[d];
15  }

// //////////////////////////////////////
// 0. Check for unknown & thresh.
// 1. Check IC
20 // 2. Check QIC
// 3. Check for at least 2 data points in this qtArr
// 4. Check for unknown to be within knowns
// 5. Sort qtArr and Return unknown conc. Move to addstandard...
// //////////////////////////////////////
25 public double getQTResult(int s, int d) {

    double m = 1.0;

    // 0. Check for unknown thresh.
30 if(!site[s].dye[d].tValid || (site[s].dye[d].dyeUsage != UNKNOWN)) {
        return 0.;
    }

    // 1. Check IC
35 if(useIC) {
        if(!site[s].dye[icDye].tValid) {
            return 0.;
        }
    }

40 // 2. Check QIC
// todo prakash.
// Should wait for all thresholds/site before constructing qtArr.
if(useQIC) {
45     if(!site[s].dye[qicDye].tValid) {
        return 0.;
    }
}

```

```

    }
    else {
        m = 1. / site[s].dye[qicDye].tCycle;
    }
5   }

    // 3. Check for at least 2 data points in this qtArr
    if(qtArr[d].length < 2) {
        return 0.;
10  }

    site[s].dye[d].conc = (float) Math.pow(10., (dyeSlope[d] *
        (site[s].dye[d].tCycle * m) + dyeOffset[d]));

15  // 4. Check for the conc to be within .5 Log
    if( (log10(site[s].dye[d].conc) > standardsLine[d][0].conc) ||
        (log10(site[s].dye[d].conc) < standardsLine[d][1].conc)) {
        site[s].dye[d].conc = 0f;
    }
20  return site[s].dye[d].conc;
}

// ////////////////////////////////////////
25  // Sort the elements in the Quantation Array.
void sort(StdElement a[]) {

    boolean done;
    StdElement se = new StdElement();

30  if(a.length < 2) {
        return;
    }

35  do {
        done = true;

        for(int j = 0; j < (a.length - 1); j++) {
            if(a[j].avgTCycle > a[j + 1].avgTCycle) {
40  done = false;
                se = a[j];
                a[j] = a[j + 1];
                a[j + 1] = se;

45  break;
            }
        }
    }

```

```

    }
    }
    while(!done);
}

5

// ////////////////////////////////////////
// Sort the elements in the Melt Peaks Array.
void sort(MeltElement meltElementsArray[]) {

10

    boolean done;
    MeltElement me = new MeltElement();

    //Debug.log ("sort: MeltElement array with " + meltElementsArray.length);
    if(meltElementsArray.length < 2) {
15
        return;
    }

    do {
20
        done = true;

        for(int j = 0; j < (meltElementsArray.length - 1); j++) {
            if(meltElementsArray[j].d1Peak > meltElementsArray[j + 1].d1Peak) {
                done = false;
25
                me = meltElementsArray[j];
                meltElementsArray[j] = meltElementsArray[j + 1];
                meltElementsArray[j + 1] = me;

                break;
30
            }
        }
    } while(!done);
}

35

// ////////////////////////////////////////
// Update data used for drawing the Line fit to standards.
//
40
// standardsLine is similar to qtArr[] but adds 2 points, one at
// conc +.5(log) and the other at conc -.5 (log).
// ////////////////////////////////////////
void updateStandards(int d) {

45
    int e = qtArr[d].length - 1;
    double conc = qtArr[d][e].conc - .5;

```

```
standardsLine[d][0].conc = qtArr[d][0].conc + .5;  
standardsLine[d][0].avgTCycle = (standardsLine[d][0].conc - dyeOffset[d])  
    / dyeSlope[d];
```

5

```
    if(conc > 0.) {  
        standardsLine[d][1].conc = conc;  
        standardsLine[d][1].avgTCycle = (conc - dyeOffset[d]) / dyeSlope[d];  
    }
```

10

```
    else {  
        standardsLine[d][1].conc = 0.;  
        standardsLine[d][1].avgTCycle = (-1 * dyeOffset[d] / dyeSlope[d]);  
    }  
}
```

15

```
// //////////////////////////////////////
```

```
// Get Control Result (Pass/Fail)
```

```
// //////////////////////////////////////
```

20

```
public boolean getControl(int s, int d) {  
    return site[s].control;  
}
```

25

```
// //////////////////////////////////////
```

```
public float getConc(int s, int d) {
```

```
    return site[s].dye[d].conc;  
}
```

30

```
// //////////////////////////////////////
```

```
public int getDyeUsage(int s, int d) {
```

```
    return site[s].dye[d].dyeUsage;  
}
```

35

```
// //////////////////////////////////////
```

```
public double getDyeSlope() {
```

```
    return dyeSlope[stdChannel];  
}
```

40

```
// //////////////////////////////////////
```

```
public double getDyeOffset() {
```

```
    return dyeOffset[stdChannel];  
}
```

45

```

// //////////////////////////////////////
// Linear Correlation Coefficient
5 public double getCC() {
    updateCC(stdChannel);

    return linCC[stdChannel];
}
10

// //////////////////////////////////////
public float getAnaData(int dataType, int s, int d, int c) {

15     float retVal = 0f;

    if (c < 0) c=0;

    switch(dataType) {

20         case NORMAL:
            if (c >=site[s].cycle) c=site[s].cycle - 1;
            if(d < 4 && d >= 0) {
                retVal = site[s].dye[d].pOptic[c];
            }
            break;

        case DERIV1:
            break;

30         case DERIV2:
            if (c >=site[s].cycle) c=site[s].cycle - 1;
            if(d < 4 && d >= 0) {
                retVal = site[s].dye[d].d2pOptic[c];
            }
            break;

35         case MELT_DERIV1:
            if (c >=site[s].meltPoints) c=site[s].meltPoints - 1;
            if(c < site[s].meltPoints && c >= 0) {
                retVal = site[s].d1mOptic.get(c);
            }
            break;

40         case MELT_OPTICS:
            if (c >=site[s].meltPoints) c=site[s].meltPoints - 1;

```

```

    if(c < site[s].meltPoints && c >= 0) {
        retVal = site[s].mOptic.get(c);
    }
    break;
5
    case MELT_TEMPERATURE:
        if (c >=site[s].meltPoints) c=site[s].meltPoints - 1;
        if(c < site[s].meltPoints && c >= 0) {
            retVal = site[s].mTemp.get(c);
10        }
            break;
        }

    return retVal;
15 }

// ////////////////////////////////////////
public int getICDye() {
20     return icDye;
}

// ////////////////////////////////////////
public boolean iCEnabled() {
25     return useIC;
}

// ////////////////////////////////////////
30 // Returns the temp assoc. with the Melt Peak.
public double getMeltTemp(int s, int index) {
    return site[s].getMeltTemp(index);
}

35
// ////////////////////////////////////////
// Returns the Melt Limit. Peak value reported only when greater.
public double getMeltLimit(int s) {
40     return site[s].meltPeakLimit;
}

// ////////////////////////////////////////
// Returns the temp assoc. with the Melt Peak.
45 public int getMeltCount(int s) {
    if (s>0 && s<numSites)

```

```

        return site[s].getMeltPeakCount();
    else
        return 0;
}
5

// ////////////////////////////////////////
public int getQICDye() {
    return qicDye;
10 }

// ////////////////////////////////////////
public boolean qicEnabled() {
    return useQIC;
15 }

// ////////////////////////////////////////
public int getTMode(int d) {
    return threshMode[d];
20 }

// ////////////////////////////////////////
int getICStartCycle() {
    return (int)validMinCycle[qicDye];
25 }

// ////////////////////////////////////////
int getICEndCycle() {
    return (int)validMaxCycle[qicDye];
30 }

// ////////////////////////////////////////
void processData(int s) {
35

    if(boxCar) {
        boxCarAvg(s);
40    }

    if(noise) {
        removeNoise(s);
45    }
}

```

```

updateThresholds(s);

// Update qtArr's. Do quantation when results are requested.
5  if(analysisType == QUANTITATIVE)
    updateQuantitative(s);
}

10  // //////////////////////////////////////
// Apply this to raw Data
void boxCarAvg(int s) {

    float sum;
15  int i;

    if(site[s].cycle < 1) {
        return;
    }

20  if(site[s].cycle + 1 >= boxCarWidth && boxCarWidth > 1) {

        for(int d = 0; d < MAX_DYES; d++) {
            sum = 0f;

25  for(i = (site[s].cycle + 1 - boxCarWidth); i < site[s].cycle + 1; i++) {
                sum += site[s].dye[d].rOptic[i];
            }

30  site[s].dye[d].pOptic[site[s].cycle] = sum / boxCarWidth;
        }
    }
}

35  // //////////////////////////////////////
void removeNoise(int s) {

    int c = site[s].cycle;
40  float temp;

    for(int d = 0; d < MAX_DYES; d++) {
        if(c >= (baselineEndCycle[d] - 1)) {

45  if(site[s].noiseValid[d]) {

```

```

site[s].dye[d].pOptic[c] -= (site[s].dye[d].slope * c + site[s].dye[d].offset);
site[s].dye[d].pOptic[c] -= site[s].dye[d].noiseAvg;

//if (s==0 && d==0) {
5 //   Logger.log("Cycle "+c+ " slope "+site[s].dye[d].slope +
//   " offset " + site[s].dye[d].offset + " pOptic " + site[s].dye[d].pOptic[c]);
//}
}
else {
10     temp = 0f;

    // Calculate Average noise
    baselineStartCycle[d] = (baselineStartCycle[d] < 1) ? 1 :
baselineStartCycle[d];
15

    site[s].dye[d].slope = 0.;
    site[s].dye[d].offset = 0.;

    site[s].dye[d].leastSquaresLineFit(baselineStartCycle[d]-1,
20 baselineEndCycle[d]-1);

    for(int i = 0; i <= (baselineEndCycle[d] - 1); i++) {
        site[s].dye[d].pOptic[i] -= (site[s].dye[d].slope * i + site[s].dye[d].offset);
    }
25

    for(int i=baselineStartCycle[d]-1; i<=baselineEndCycle[d]-1; i++) {
        temp = temp + site[s].dye[d].pOptic[i];
    }

30     site[s].dye[d].noiseAvg = temp / (baselineEndCycle[d] -
baselineStartCycle[d] + 1);

    // Remove noise
    for(int i=0; i <= (baselineEndCycle[d]-1); i++) {
35         site[s].dye[d].pOptic[i] -= site[s].dye[d].noiseAvg;
    }
    site[s].noiseValid[d] = true;
}
}
}
40 }

// ////////////////////////////////////////
45 void updateThresholds(int s) {

```

```

for(int d = 0; d < MAX_DYES; d++) {

    // Update Derivative
    update2D(s, d);

5   if(dataType[d] == PRIMARY) {
        if(threshMode[d] == MAN_THRESH) {
            updateThreshPDMan(s, d);
        }
        else {
10         updateThreshPDAuto(s, d);
        }
    }
    else {
15         if(threshMode[d] == MAN_THRESH) {
            updateThresh2DMan(s, d);
        }
        else {
            updateThresh2DAuto(s, d);
20         }
        }
    }
}

25 // //////////////////////////////////////
int updateThreshPDMan(int s, int d) {

    int c = site[s].cycle;
    int du = site[s].dye[d].dyeUsage;

30     if(du == UNUSED || du == PASSIVE) {
        return 0;
    }

35     if(noise) {
        if(c <= baselineEndCycle[d]) {
            return 0;
        }
    }

40 }

    if(!site[s].dye[d].tValid) {
        if(site[s].dye[d].pOptic[c] >= site[s].dye[d].tLimit) {

45         // Optic exceeded limit, calculate cycle
        if(c >= 1) {

```

```

    site[s].dye[d].tValid = true;

    LinearFit l;

5    l = new LinearFit(c - 1, site[s].dye[d].pOptic[c - 1], c,
        site[s].dye[d].pOptic[c]);

    // zero based
    site[s].dye[d].tCycle = l.fitY(site[s].dye[d].tLimit) + 1f;
10    }
    }
    }
    return 0;
}

15

// //////////////////////////////////////
// When not to find the Threshold crossing:
//
20 // 1. Unused Dye
// 2. Passive dye
// 3. Already found (.tValid)
// 4. Not enough cycles (2D)
// 5. All dyes don't have valid stdDev Auto
25 // //////////////////////////////////////
int updateThreshPDAuto(int s, int d) {

    int c = site[s].cycle;
    float sum, temp;
30    int du = site[s].dye[d].dyeUsage;

    if(du == UNUSED || du == PASSIVE) {
        return 0;
    }

35    if(c <= baselineEndCycle[d]) {
        return 0;
    }

40    if(maxStdDevValid[d] &&!site[s].dye[d].tValid) {

        // Look for signal crossing
        if(site[s].dye[d].pOptic[c] > site[s].dye[d].tLimit) {

45            LinearFit l;

```

```

l = new LinearFit(c - 1, site[s].dye[d].pOptic[c - 1], c, site[s].dye[d].pOptic[c]);

// Add one to match graph
site[s].dye[d].tCycle = l.fitY(site[s].dye[d].tLimit) + 1.0f;
5   site[s].dye[d].tValid = true;
    }
  }
  else if(!maxStdDevValid[d] &&!site[s].dye[d].tValid) {

10   // If enough data, calculate stdDev
    // No need to check crossing yet.
    if(c >= baselineEndCycle[d]) {
      if((baselineEndCycle[d] - baselineStartCycle[d]) > 1) {

15       // mean
        sum = 0f;

        for(c = (baselineStartCycle[d] - 1); c <= (baselineEndCycle[d] - 1); c++) {
          sum = sum + site[s].dye[d].pOptic[c];
20       }

        site[s].dye[d].mean = sum / (baselineEndCycle[d] - baselineStartCycle[d] +
1)1);

25       // stdDev
        sum = 0f;

        for(c = (baselineStartCycle[d] - 1); c <= (baselineEndCycle[d] - 1); c++) {
          temp = site[s].dye[d].pOptic[c] - site[s].dye[d].mean;
          sum = sum + temp * temp;
30       }

        site[s].dye[d].stdDev = (float) Math.sqrt(sum / (baselineEndCycle[d] -
baselineStartCycle[d]));
35       site[s].dye[d].stdDevValid = true;

        setMaxStdDev(d);
      }
    }
40   }

  return 0;
}

45 // //////////////////////////////////////

```

```

// This function calculates the Cycle Threshold for Primary Data with
// a manual threshold limit set by the user.
// //////////////////////////////////////
5  int updateThresh2DMan(int s, int d) {

    int du = site[s].dye[d].dyeUsage;

    // Because the calculation for D2 is lagging 2 cycles back.
    int c = site[s].cycle - 2;

10  if(du == UNUSED || du == PASSIVE) {
        return 0;
    }

15  if(c < 6) {
        return 0;
    }

    if(noise) {
20      if(c <= baselineEndCycle[d]) {
          return 0;
        }
    }

25  // Look for peak
    // When c == 6, Possible valid D2's are at c2(c-4), c3(c-3), c4(c-2)
    if((site[s].dye[d].d2pOptic[c - 3] > site[s].dye[d].d2pOptic[c - 4]) &&
        (site[s].dye[d].d2pOptic[c - 3] >= site[s].dye[d].d2pOptic[c - 2])) {

30      PeakFinder peakFinder = new PeakFinder((float) (c - 4),
        site[s].dye[d].d2pOptic[c - 4],
        (float) (c - 3), site[s].dye[d].d2pOptic[c - 3], (float) (c - 2),
        site[s].dye[d].d2pOptic[c - 2]);

35      // Look for signal crossing
        if(peakFinder.peak > site[s].dye[d].tLimit) {

            // peak exceeded limit, calculate cycle
            // Note: peak is 3 cycles back from here
40            if(site[s].dye[d].tValid) {

                if (site[s].dye[d].tCycle < peakFinder.cycle + 1.0f) {
                    site[s].dye[d].tCycle = peakFinder.cycle + 1.0f;
                }

45        }
    }
}

```

```

        else {
            site[s].dye[d].tValid = true;
            site[s].dye[d].tCycle = peakFinder.cycle + 1.0f;
        }
5      }
    }
    return 0;
}

10
// //////////////////////////////////////
// //////////////////////////////////////
int updateThresh2DAuto(int s, int d) {

15    int du = site[s].dye[d].dyeUsage;
    float sum, temp;
    int cy;

    // Because the calculation for D2 is lagging 2 cycles back.
20    int c = site[s].cycle - 2;

    if(du == UNUSED || du == PASSIVE) {
        return 0;
    }

25    if(c < 6) {
        return 0;
    }

    if(c <= baselineEndCycle[d]) {
30        return 0;
    }

    if(maxStdDevValid[d]) {

35        // Look for signal crossing, ie Look for peak
        // When c == 6, Possible valid D2's are at c2(c-4), c3(c-3), c4(c-2)
        if(c < (baselineEndCycle[d] + 3)) {
            return 0;
40        }

        if((site[s].dye[d].d2pOptic[c - 3] >= site[s].dye[d].d2pOptic[c - 4]) &&
            (site[s].dye[d].d2pOptic[c - 3] > site[s].dye[d].d2pOptic[c - 2])) {

45            PeakFinder m = new PeakFinder((float) (c - 4), site[s].dye[d].d2pOptic[c - 4],
                (float) (c - 3), site[s].dye[d].d2pOptic[c - 3], (float) (c - 2),

```

```

    site[s].dye[d].d2pOptic[c - 2]);

// Look for signal crossing
if(m.peak > site[s].dye[d].tLimit) {
5
    if (site[s].dye[d].tValid) {
        if (site[s].dye[d].tCycle < m.cycle + 1f) {
            site[s].dye[d].tCycle = m.cycle + 1f;
        }
10
    }
    else {
        // peak exceeded limit, calculate cycle
        site[s].dye[d].tValid = true;
        site[s].dye[d].tCycle = m.cycle + 1f;
15
    }
}
}
}
else if(!maxStdDevValid[d] &&!site[s].dye[d].tValid) {
20
    // If enough data, calculate stdDev
    // No need to check crossing yet.
    if(c >= baselineEndCycle[d]) {
        if((baselineEndCycle[d] - baselineStartCycle[d]) > 1) {
25
            // mean
            sum = 0f;

            for(c = (baselineStartCycle[d] - 1); c <= (baselineEndCycle[d] - 1); c++) {
30
                sum = sum + site[s].dye[d].d2pOptic[c];
            }

            // Changed 1/12/00 as per SCR 129.
            // sum = sum + site[s].dye[d].pOptic[c];
35

            site[s].dye[d].mean = sum / (baselineEndCycle[d] - baselineStartCycle[d] +
1);

            // stdDev
            sum = 0f;
40

            for(c = (baselineStartCycle[d] - 1); c <= (baselineEndCycle[d] - 1); c++) {

                // Changed 1/12/00 as per SCR 129.
45
                // temp = site[s].dye[d].pOptic[c] - site[s].dye[d].mean;

```

```

        temp = site[s].dye[d].d2pOptic[c] - site[s].dye[d].mean;
        sum = sum + temp * temp;
    }

5    site[s].dye[d].stdDev = (float) Math.sqrt(sum / (baselineEndCycle[d] -
baselineStartCycle[d]));
    site[s].dye[d].stdDevValid = true;

    setMaxStdDev(d);
10    }
    }
    }

    return 0;
15    }

```

```

// ////////////////////////////////////////
// Update 2nd Deriv for optic data
20 // ////////////////////////////////////////
void update2D(int s, int d) {

    int c = site[s].cycle;
    float mult = 6.25f;

25    if (c<4)
        return;

    // D2
30    if(c < MAX_CYCLES - 1 && c > 2) {

        // n=3 thru n-2

        /*
35    //float mult = 5f;
    site[s].dye[d].d2pOptic[c - 2] = (site[s].dye[d].arD1Dye[c - 1] -
        site[s].dye[d].arD1Dye[c - 3]) / 2f * mult;
    site[s].dye[d].d2pOptic[c - 1] = (site[s].dye[d].arD1Dye[c] -
        site[s].dye[d].arD1Dye[c - 2]) / 2f * mult;
40    site[s].dye[d].d2pOptic[c] = (site[s].dye[d].arD1Dye[c] -
        site[s].dye[d].arD1Dye[c - 1]) * mult;

    */
    site[s].dye[d].d2pOptic[c-2] = (site[s].dye[d].pOptic[c] -
        2f * site[s].dye[d].pOptic[c-2] +
45    site[s].dye[d].pOptic[c-4]) * mult;

```

```

    site[s].dye[d].d2pOptic[c-1] = (2f * site[s].dye[d].pOptic[c] -
        3f * site[s].dye[d].pOptic[c-1] +
        site[s].dye[d].pOptic[c-3]) * mult;

5    site[s].dye[d].d2pOptic[c] = (site[s].dye[d].pOptic[c] -
        2f * site[s].dye[d].pOptic[c-1] +
        site[s].dye[d].pOptic[c-2]) * 2 * mult;
    }
    else {
10    site[s].dye[d].d2pOptic[c] = 0f;
    }
}

15 // //////////////////////////////////////
// Update qtArr's (1 per dye - site independent).
// Only if std: only with valid thresh
// //////////////////////////////////////
void updateQuantitative(int s) {
20    for(int d = 0; d < MAX_DYES; d++) {
        if(site[s].dye[d].dyeUsage == STANDARD) {
            // if(site[s].dye[d].tValid) {

25            if( (useQIC && (getTCycle(s, qicDye) > 0f)) || getTCycle(s, d) > 0f ) {
                addStandard(s, d);
                //updateStandards(d);
                LeastSquares ls = new LeastSquares(qtArr[d], d);
                dyeSlope[d] = ls.getSlope();
                dyeOffset[d] = ls.getOffset();
30                updateStandards(d);
            }
        }
    }
35 }

// //////////////////////////////////////
//
40 // Add a stdElement to the qlArr if appropriate.
// If QIC used - valid
// If IC used - valid
// Sort if more than 1 element
// //////////////////////////////////////
45 int addStandard(int s, int d) {
    int i;

```

```

float tCycle;

if(!site[s].dye[d].tValid || getTCycle(s,d) <= 0f ) {
    return 0;
5  }

if(site[s].dye[d].conc < 10E-5f) {
    return 0;
}

10  if (useQIC) {
    tCycle = getQICTCycle(s,d);
    }
    else {
15     tCycle = getTCycle(s,d);
    }

if (qtArr[d][0].conc < -9) {
    // Initialise
20     qtArr[d][0].conc = log10(site[s].dye[d].conc);
    qtArr[d][0].avgTCycle = tCycle;
    qtArr[d][0].nElements = 1;
    return 0;
}

25  else {

    // Look for conc in array
    for(i = 0; i < qtArr[d].length; i++) {

30     if(Math.abs(qtArr[d][i].conc - log10(site[s].dye[d].conc)) < .05) {
        qtArr[d][i].avgTCycle = ((qtArr[d][i].avgTCycle * qtArr[d][i].nElements) +
                                tCycle) / (qtArr[d][i].nElements + 1);
        qtArr[d][i].nElements += 1;

35     // May need to be resorted
        if(qtArr[d].length > 1) {
            sort(qtArr[d]);
        }

40     return 0;
    }
}

// Conc not found, add new element to array
45  StdElement tempArr[] = new StdElement[qtArr[d].length + 1];

```

```

// Initialise tempArr
for(i = 0; i < tempArr.length; i++) {
    tempArr[i] = new StdElement();
}

5   System.arraycopy(qtArr[d], 0, tempArr, 0, qtArr[d].length);

    tempArr[tempArr.length - 1].conc = log10(site[s].dye[d].conc);
    tempArr[tempArr.length - 1].avgTCycle = tCycle;
10   tempArr[tempArr.length - 1].nElements = 1;
    qtArr[d] = tempArr;

    // Sort
    sort(qtArr[d]);
15   }

    return 0;
}

20   ///////////////////////////////////////////////////////////////////
void setMaxStdDev(int d) {

    maxStdDevValid[d] = true;

25   int s;

    maxStdDev[d] = 0f;

30   for(s = 0; s < numSites; s++) {
        if(site[s].dye[d].stdDevValid) {
            if(site[s].dye[d].stdDev > maxStdDev[d]) {
                maxStdDev[d] = site[s].dye[d].stdDev;
            }
        }
35   }
        else {
            maxStdDevValid[d] = false;
            maxStdDev[d] = 0f;

40   return;
        }
    }

45   if(maxStdDevValid[d]) {

        // All sites have stdDevValid for dye d,

```

```

// Calculate Threshold limits
for(s = 0; s < numSites; s++) {
    site[s].dye[d].tLimit = (float)(stdDevBaseLine[d] * maxStdDev[d]);
    //System.out.println("stdDevBaseLine[d] " + stdDevBaseLine[d] +
5    // "maxStdDev[d] " + maxStdDev[d] +
    // " setMaxStdDev " + site[s].dye[d].tLimit );
}
}
}
10

// ////////////////////////////////////////
public Analysis() {
    this(MAX_SITES);
15 }

public Analysis(int ns) {

    numSites = ns;

20    site = new Site[numSites];

    for(int i = 0; i < numSites; i++) {
        site[i] = new Analysis.Site();
25    }

    analysisType = QUALITATIVE;

    useQIC = false;
30    qicDye = 0;
    useIC = false;
    icDye = 0;

    boxCar = false;
35    boxCarWidth = 0;

    // Default to match noise sub with primary data.
    // noise = false;

40    for(int i = 0; i < MAX_DYES; i++) {
        threshMode[i] = AUTO_THRESH;
        stdDevBaseLine[i] = 5f;
        maxStdDev[i] = 0f;
        maxStdDevValid[i] = false;
45        dataType[i] = PRIMARY;
        qtArr[i][0] = new StdElement();
    }
}

```

```

baselineStartCycle[i] = 3;
baselineEndCycle[i] = 8;

// Standards Curve, prakash 1/25/00
5 standardsLine[i][0] = new StdElement();
  standardsLine[i][1] = new StdElement();

// Optics must cross threshold in this range
10 validMinCycle[i] = 3f;
   validMaxCycle[i] = 60f;
   }
 }

15 ///////////////////////////////////////////////////////////////////
   class Site {

       Dye dye[] = new Dye[MAX_DYES];

20 // Melt Peak Analysis
   private Array.Short mOptic = new Array.Short(32);
   private Array.Float mTemp = new Array.Float(32);
   private Array.Float d1mOptic = new Array.Float(32);
   private MeltElement mPeaks[] = new MeltElement[1];

25 // Possible to set per site in future.
   private double meltPeakLimit = 10.;

// Melt peaks processed
30 private boolean meltPeaksValid;

// Current Cycle Number
   int cycle;

35 // Number of MeltData points
   private int meltPoints;

// IC/QIC passed:T; failed:F
   boolean control;

40 // Noise
   boolean noiseValid[] = new boolean [MAX_DYES];

   Site() {

45 // Initialise dyes

```

```

    for(int i = 0; i < MAX_DYES; i++) {
        dye[i] = new Dye();
        noiseValid[i] = false;
    }

5   cycle = 0;
    meltPoints = 0;
    meltPeaksValid = false;
    control = false;
10   mPeaks[0] = new MeltElement();
    }

    private void updateMeltDeriv() {

15     meltPeaksValid = false;

        if(meltPoints < 1) {
            d1mOptic.set(0, 0f);
        }
20     else if(meltPoints == 1) {
            d1mOptic.set(1, (mOptic.get(1) - mOptic.get(0)) * -5f);
        }
        else {
            // Recalc the 2nd last value, and the last value
25     d1mOptic.set(meltPoints-1, (mOptic.get(meltPoints) -
mOptic.get(meltPoints-2)) / 2f * -5f);
            d1mOptic.set(meltPoints, (mOptic.get(meltPoints) -
mOptic.get(meltPoints-1)) * -5f);
30     }
        }

        // Return number of Melt Peaks detected.
        private int getMeltPeakCount() {
            if (!meltPeaksValid)
35             detectMeltPeaks();
            return (mPeaks[0].temp < 0.) ? 0 : mPeaks.length;
        }

        // Return number of Melt Temp Associated with Peak.
40     private double getMeltTemp(int index) {
            if (index < getMeltPeakCount())
                return mPeaks[index].temp;
            else
45             return 0f;
        }
    }

```

```
// Find all peaks in 1st Deriv of Melt Optic
private void detectMeltPeaks() {
```

```
    if (meltPoints < 2) return;
```

5.

```
    if (!meltPeaksValid) {
```

```
        meltPeaksValid = true;
```

```
        mPeaks = new MeltElement[1];
```

```
        mPeaks[0] = new MeltElement();
```

10

```
        // Debug.log("detectMP, length " + mPeaks.length);
```

```
        for (int i=1; i<meltPoints-1; i++) {
```

15

```
            if( ( d1mOptic.get(i) > d1mOptic.get(i-1) ) &&
                ( d1mOptic.get(i) >= d1mOptic.get(i+1) ) ) {
```

```
                PeakFinder peakFinder = new PeakFinder((float)(i-1),
                (float)d1mOptic.get(i-1),
                (float)(i), (float)d1mOptic.get(i), (float)(i+1),
                (float)d1mOptic.get(i+1));
```

20

```
                // Look for signal crossing
```

```
                if (peakFinder.peak > meltPeakLimit) {
```

25

```
                    if (mPeaks[0].temp < 0.) {
```

```
                        mPeaks[0].d1Peak = peakFinder.peak;
```

```
                        mPeaks[0].temp = mTemp.get(0) + peakFinder.cycle; // Temp,
```

in this case.

```
                    }
```

30

```
                    else {
```

```
                        MeltElement tempA[] = new MeltElement[mPeaks.length+1];
```

```
                        // Initialise tempA
```

```
                        for(int j = 0; j < tempA.length; j++) {
```

35

```
                            tempA[j] = new MeltElement();
```

```
                        }
```

```
                        System.arraycopy(mPeaks, 0, tempA, 0, mPeaks.length);
```

40

```
                        tempA[tempA.length-1].d1Peak = peakFinder.peak;
```

```
                        tempA[tempA.length-1].temp = mTemp.get(0) +
```

peakFinder.cycle; // Temp, in this case.

```
                        mPeaks = tempA;
```

```
                    }
```

45

```
                }
```

```
            }
```

```

    }
}

//Debug.log(" detectMeltPeaks() mPeaks.length " + mPeaks.length);
5   if (mPeaks.length > 1)
        sort(mPeaks);
    }
}

10  ///////////////////////////////////////////////////////////////////
    class Dye {

        // Data Arrays
        short rOptic[] = new short[MAX_CYCLES];
15   float pOptic[] = new float[MAX_CYCLES];

        // 2nd derivative
        float d2pOptic[] = new float[MAX_CYCLES];

20   // Threshold limit
        float tLimit;
        float tCycle;

        // Indicates if signal crossed the Threshold Limit
25   boolean tValid;

        // Qualitative Result
        int qlResult;

30   // IC, QIC, Unused, ...
        int dyeUsage;

        // true = Std; false = Unkn
        boolean std;

35   // Dye Concentration
        float conc;

        // Background Noise Value
40   float noiseAvg;

        // Std Dev, Mean calculated. one per dye per site
        boolean stdDevValid;
        float stdDev;
45   float mean;

```

```
// For slope removal. One per dye per site
double slope;
double offset;
```

```
5 Dye() {
```

```
    // Initialise arrays
    for(int i = 0; i < MAX_CYCLES; i++) {
        rOptic[i] = 0;
10        pOptic[i] = 0f;
        d2pOptic[i] = 0f;
    }
```

```
15    // Default Man Threshold, dyeUsage, tValid
    qlResult = 0;
    tLimit = 200f;
    tCycle = 0f;
    tValid = false;
    dyeUsage = ASSAY;
20    std = false;
    conc = 10E-6f;
    noiseAvg = 0f;
    stdDevValid = false;
    stdDev = 0f;
25    mean = 0f;
    slope = 0.;
    offset = 0.;
}
```

```
30 void endPointLineFit(int start, int end) {
    slope = (pOptic[end] - pOptic[start]) / (double)(end - start);

    if ((slope * end) != 0.) {
        offset = pOptic[end] / (slope * end);
35    }
    else {
        offset = 0.;
    }
}
```

```
40 void leastSquaresLineFit(int start, int end) {

    if ((end - start) < 2) {
        return;
45    }
    LeastSquares ls = new LeastSquares(pOptic, start, end);
```

```

    slope = ls.getSlope();

    if ((slope * end) != 0.) {
        offset = ls.getOffset();
    }
    else {
        offset = 0.;
    }
}
}

```

```

// //////////////////////////////////////
public class StdElement {
    public double conc;
    public double avgTCycle;
    int nElements;

    StdElement() {
        conc = -10.;
        avgTCycle = 0.;
        nElements = 0;
    }
}

```

```

// //////////////////////////////////////
public class MeltElement {
    public double temp = -1.;
    public double d1Peak = -1.;
}

```

```

// //////////////////////////////////////
// //////////////////////////////////////
public static void main(String args[]) {

    int s, d, c, cy;
    Analysis a = new Analysis();

    // For reading data from Excel
    Vector vFam = new Vector(16);
    vFam.setSize(16);
    Vector vTet = new Vector(16);
    vTet.setSize(16);
    Vector vTam = new Vector(16);
}

```

```

vTam.setSize(16);
Vector vRox = new Vector(16);
vRox.setSize(16);

5 // Analysis Type
a.setAnalysisType(QUALITATIVE);
//a.setAnalysisType(QUANTITATIVE);

a.setNumSites(16);

10

for (d=0; d<MAX_DYES; d++) {

//a.setDataTyp(d, D2); // Set Up Data Type
15 a.setDataTyp(d, PRIMARY);

a.threshMode[d] = AUTO_THRESH; // Set Thresh Mode
//a.threshMode[d] = MAN_THRESH;

20 a.stdDevBaseLine[d] = 5.;
}

// Set Threshold
//a.setTLimit(0, 10f);
25 //a.setTLimit(1, 10f);
//a.setTLimit(2, 10f);
//a.setTLimit(3, 10f);

// Test BoxCar Avg
30 a.setBoxCarAvg(true, 3);

// Test QIC Dye
a.setDyeUsage(0, 1, QIC);

35 // Test Background Noise Subtraction
a.setNoiseSubtraction(true);

// Valid Min, Max Cycle defaults to 3, 60
//a.setICCycle(3, 30, 60);

40 // Add Data Thresholds and cycle crossings are calculated as soon as
// enough data has accumulated.

45 try {

```

```
BufferedReader in = new BufferedReader(new  
FileReader("data5.csv"));
```

```
String str;
```

5

```
// Throw away first 2 lines
```

```
str = in.readLine();
```

```
str = in.readLine();
```

10

```
while ((str = in.readLine()) != null) {
```

```
    //Debug.log(str.length()+" "+ str);
```

```
    StringTokenizer t = new StringTokenizer(str, ",");
```

```
    for (int i=0; i<16; i++)
```

15

```
        if (t.hasMoreTokens())
```

```
            vFam.setElementAt( (Integer.valueOf(t.nextToken())), i);
```

```
    for (int i=0; i<16; i++)
```

```
        if (t.hasMoreTokens())
```

20

```
            vTet.setElementAt((Integer.valueOf(t.nextToken() )), i );
```

```
    for (int i=0; i<16; i++)
```

```
        if (t.hasMoreTokens())
```

```
            vTam.setElementAt((Integer.valueOf(t.nextToken() )), i );
```

25

```
    for (int i=0; i<16; i++)
```

```
        if (t.hasMoreTokens())
```

```
            vRox.setElementAt((Integer.valueOf(t.nextToken() )), i );
```

30

```
        for (s=0; s<16; s++) {
```

```
            Integer aa = (Integer)vFam.elementAt(s);
```

```
            Integer bb = (Integer)vTet.elementAt(s);
```

```
            Integer cc = (Integer)vTam.elementAt(s);
```

35

```
            Integer dd = (Integer)vRox.elementAt(s);
```

```
            a.addCycle(s, aa.shortValue(), bb.shortValue(),  
cc.shortValue(), dd.shortValue() );
```

40

```
            // cy = a.site[s].cycle -1;
```

```
            //Debug.log("Main: Site " +s+ " Cycle " +cy+ " " +
```

```
a.site[s].dye[0].rOptic[cy]+
```

```
            // " "+a.site[s].dye[1].rOptic[cy]+
```

```
            // " "+a.site[s].dye[2].rOptic[cy]+
```

45

```
            // " "+a.site[s].dye[3].rOptic[cy] );
```

```
        }
```

```

    }
    }
    catch(IOException e) {
5       Debug.log("IOException");
    }

    // Set up Melt Inverse of FAM
    for (s=0; s<16; s++) {
10       for (short sec=0; sec<a.site[s].cycle; sec++) {
            //Debug.log ("Adding data to Melt " + sec + " " +
            a.site[s].dye[1].rOptic[sec]);
            a.addMelt(s, sec, a.OPTICS, a.site[s].dye[1].rOptic[sec]);
            a.addMelt(s, sec, a.TEMP, (short)(60+sec));
15       }
    }

    /*
    // Set UP for quantation.
    // 100
    a.setSiteType(0, SITE_STANDARD);
    a.setConc(0, 0, 100f);

    a.setSiteType(1, SITE_STANDARD);
    a.setConc(1, 0, 100f);

    //1000
    a.setSiteType(3, SITE_STANDARD);
    a.setConc(3, 0, 1000f);

    a.setSiteType(8, SITE_STANDARD);
    a.setConc(8, 0, 1000f);

    //10
    a.setSiteType(14, SITE_STANDARD);
    a.setConc(14, 0, 10f);

    a.setSiteType(15, SITE_STANDARD);
    a.setConc(15, 0, 10f);

    // Unknowns
    a.setSiteType(2, SITE_UNKNOWN);
    a.setSiteType(4, SITE_UNKNOWN);
    a.setSiteType(5, SITE_UNKNOWN);
    a.setSiteType(6, SITE_UNKNOWN);
    a.setSiteType(7, SITE_UNKNOWN);

```

```

5      for (int i=9; i<14; i++)
        a.setSiteType(i, SITE_UNKNOWN);
      */

      /*
      // Force QIC Cycle for testing
      for (int i=0; i<16; i++) {
10         a.setTCycle(i, 1, (float)(10+.1*i));
        //a.setTCycle(i, 1, 10f );
        a.site[i].dye[1].tValid = true;
      }

      for(int i=0; i<a.numSites; i++)
15         a.updateQuantitative(i);
      */

      // (site, dye, data)
      //a.dLog(7, 1, 1); // outputs threshold limits + Cycle num
      //a.dLog(7, 0, 0); // outputs data
      //a.dLog(7, 1, 2); // outputs raw + 2d
      //a.dLog(7, 0, 3); // outputs threshold limits + Cycle num
      //a.dLog(7, 0, 4); // outputs threshold limits + Cycle num + QIResult
      //a.dLog(0, 0, 5); // outputs Tlimits + TCycle num + conc (dye, all
20
      sites)
        //a.dLog(0, 0, 6); // outputs qtArr for given dye
        //a.dLog(7, 1, 7); // outputs threshold limits + Cycle num + QIC
      Cycle numbers
        //a.dLog(7, 1, 8); // Outputs melt data for given site.
        //a.dLog(7, 1, 9); // Outputs melt data peaks for given site.
30

      Debug.log("*****");
      Debug.log("data4.csv, primary w Man Thresh,
      setNoiseSubtraction(true));
      Debug.log("setBoxCarAvg(true, 3) Quantitative ");
35      Debug.log("*****");
      a.dLog(3, 0, 2);
    }

40
    //////////////////////////////////////
    //////////////////////////////////////
    // Used for unit testing
    void dLog(int st, int dy, int data) {
45
        int i, s, d, c;

```

```

switch (data) {
case 0:
    // data
    5    Debug.log("dLog: pOptic 7,* - Cy 0-44");

    for (i=0; i<site[st].cycle; i++)
        Debug.log(" " + site[st].dye[0].pOptic[i] +
            " " + site[st].dye[1].pOptic[i] +
            " " + site[st].dye[2].pOptic[i] +
            " " + site[st].dye[3].pOptic[i] );
    10    break;

    case 1:
        // thresh Limits, Cycle Numbers
        15    for (s=0; s<numSites; s++)
            for (d=0; d<MAX_DYES; d++)
                Debug.log("Site " + s +
                    " Dye " + d +
                    " Thresh " + getTLimit(s, d) +
                    " Cycle " + getTCycle(s, d) );
            20    break;

        // Prints raw + 2d data for st, dy
        case 2:
            25    for (c=0; c<site[st].cycle; c++)
                Debug.log("Site " + st +
                    " Dye " + dy +
                    " Cycle " + c +
                    " raw data " + site[st].dye[dy].rOptic[c] +
                    " data " + site[st].dye[dy].pOptic[c] +
                    " 2D " + site[st].dye[dy].d2pOptic[c] );
            30    break;

        // Prints dy channel TCycles and TLimits
        case 3:
            35    for (s=0; s<numSites; s++)
                Debug.log("Site " + s +
                    " Dye " + dy +
                    " Thresh Cycle " + getTCycle(s, dy) +
                    " Thresh Limit " + getTLimit(s, dy)
                    40    );
                break;

        // Prints dy channel TCycles and TLimits and QI Results
        case 4:
            45    for (s=0; s<numSites; s++)

```

```

5      Debug.log("Site " + s +
        " Dye " + dy +
        " Thresh Cycle " + getTCycle(s, dy) +
        " Thresh Limit " + getTLimit(s, dy) +
        " Result " + getQLResult(s, dy)
        );
        break;

10     // Prints dy channel TCycles and Qn Results
        // for dye at all sites
        case 5:
        for (s=0; s<numSites; s++)
            if (useQIC) {
15                Debug.log("Site " + s +
                    " Dye " + dy +
                    " QIC Thresh Cycle " + getQICTCycle(s, dy) +
                    " Result " + getQTRResult(s, dy)
                    );
            }
            else {
20                Debug.log("Site " + s +
                    " Dye " + dy +
                    " Thresh Cycle " + getTCycle(s, dy) +
                    " Result " + getQTRResult(s, dy)
                    );
            }
            break;

25     case 6:
        for (c=0; c<qtArr[0].length; c++)
30            Debug.log(" qtArr[0] Len " + qtArr[0].length + " conc " +
                qtArr[0][c].conc + " Avg cy " + qtArr[0][c].avgTCycle);
            break;

35     // Prints dy channel TCycles and TLimits + QIC
        case 7:
        for (s=0; s<numSites; s++) {
            for (dy=0; dy<4; dy++) {
40                Debug.log("Site " + s +
                    " Dye " + dy +
                    " Thresh Cycle " + getTCycle(s, dy) +
                    " QIC Thresh Cycle " + getQICTCycle(s, dy) +
                    " Thresh Limit " + getTLimit(s, dy)
                    );
            }
45        }
    }

```

break;

// Prints melt for given site

case 8:

5

for (c=0; c<site[st].cycle; c++) {

Debug.log("Site " + st +

" sec " + c +

" mOptic " + site[st].mOptic.get(c) +

" d1mOptic " + site[st].d1mOptic.get(c) +

10

" Temp " + site[st].mTemp.get(c)

);

}

break;

15

// Prints melt Peaks for given site

case 9:

for (c=0; c<site[st].getMeltPeakCount(); c++) {

Debug.log("Site " + st +

" MeltPoint " + c +

20

" d1peak " + site[st].mPeaks[c].d1Peak +

" temp " + getMeltTemp(st, c)

);

}

break;

25

}

}

}

```

// //////////////////////////////////////
// Least Squares Fit. Takes an array of points (x,y pairs) and calculates
// the slope and offset using the 'Least Squares Fit' method.
// //////////////////////////////////////
5  class LeastSquares {

    double sumX = 0.;
    double sumY = 0.;
    double sumXY = 0.;
10  double sumOfXSq = 0.;
    double sumXSquared = 0.;
    int arrayLen = 0;
    double slope = 0.;

15  LeastSquares() {};

    // Used for quantation.
    LeastSquares(Analysis.StdElement a[], int d) {

20      arrayLen = a.length;

      for(int i = 0; i < arrayLen; i++) {
        sumX += a[i].avgTCycle;
        sumY += a[i].conc;
25      sumXY += a[i].avgTCycle * a[i].conc;
        sumOfXSq += a[i].avgTCycle * a[i].avgTCycle;
      };
      sumXSquared = sumX * sumX;
    };

30  // Used for removing background noise

```

```
LeastSquares(float optic[], int start, int end) {
```

```
    arrayLen = end - start + 1;
```

```
5    for(int i = start; i < end+1; i++) {
```

```
        sumX += i;
```

```
        sumY += optic[i];
```

```
        sumXY += i * optic[i];
```

```
        sumOfXSq += i * i;
```

```
10    }
```

```
    sumXSquared = sumX * sumX;
```

```
};
```

```
double getSlope() {
```

```
15    if(Math.abs(sumOfXSq - sumXSquared / arrayLen) > 10E-10) {
```

```
        slope = (sumXY - (sumY * sumX / arrayLen)) /  
                (sumOfXSq - (sumXSquared / arrayLen));
```

```
    }
```

```
    else {
```

```
20        slope = 0.;
```

```
    }
```

```
    return slope;
```

```
}
```

```
25 double getOffset(){
```

```
    return (sumY / arrayLen) - (slope * sumX / arrayLen);
```

```
}
```

```
}
```

```

// //////////////////////////////////////
// This object takes 2 points (x,y) pairs and calculates the slope and
// offset. It returns the unknown (either x or y) using the equation
//  $y = mx + b$ .
5 // //////////////////////////////////////
class LinearFit {

    double m;
    double b;

10 LinearFit() {};

    LinearFit(int x1, double y1, int x2, double y2) {
        m = 0.;
15 b = 0.;

        if((x1 - x2) != 0) {
            m = (y1 - y2) / (x1 - x2);
            b = y1 - m * x1;
20 }
        }

    LinearFit(float x1, double y1, float x2, double y2) {
        m = 0.;
25 b = 0.;

        if((x1 - x2) != 0) {
            m = (y1 - y2) / (x1 - x2);
            b = y1 - m * x1;
30 }
        }
}

```

```
float fitX(float x) {  
    return (float) (m * x + b);  
}
```

5

```
float fitY(float y) {  
    if(m != 0) {  
        return (float) ((y - b) / m);  
    }  
    else {  
        return 0;  
    }  
}
```

10

15

```

// //////////////////////////////////////
// Determines the Peak and Cycle for the second derivative. It takes 3
// points (x,y pairs) and fits a line of the 2nd order through all three
// points. peak(y) is optic and cycle(x) is the PCR Cycle number.
5 // //////////////////////////////////////
class PeakFinder {

    float peak;
    float cycle;
10    double d0, d1, d2, d3;
    double r1, r2, r3;

    PeakFinder () {};

15    PeakFinder(float x1, float y1, float x2, float y2, float x3, float y3) {
        d0 = det((x1 * x1), x1, 1, (x2 * x2), x2, 1, (x3 * x3), x3, 1);

        d1 = det(y1, x1, 1, y2, x2, 1, y3, x3, 1);

20        d2 = det((x1 * x1), y1, 1, (x2 * x2), y2, 1, (x3 * x3), y3, 1);

        d3 = det((x1 * x1), x1, y1, (x2 * x2), x2, y2, (x3 * x3), x3, y3);

        if(d0 != 0f) {
25            r1 = d1 / d0;
            r2 = d2 / d0;
            r3 = d3 / d0;
            cycle = (float) ((-1 * r2) / (2 * r1));
            peak = (float) (r3 - (r2 * r2) / (4 * r1));
30        }
        else {

```

cycle = 0f;

peak = 0f;

}

}

5

//

double det(float a11, float a12, float a13, float a21, float a22, float a23,  
float a31, float a32, float a33) {

10

return ( (a11 \* a22 \* a33) + (a12 \* a23 \* a31) + (a13 \* a21 \* a32) -  
(a31 \* a22 \* a13) - (a32 \* a23 \* a11) - (a33 \* a21 \* a12));

}

}

15